ОБЩЕСТВО «ЗНАНИЕ» РСФСР

Ленинградская организация

А. Ю. БУРАГО;
В. А. КИРИЛЛИН,
кандидат физико-математических наук;
И. В. РОМАНОВСКИЙ,
доктор физико-математических наук

ФОРТ — ЯЗЫК ДЛЯ МИКРОПРОЦЕССОРОВ

(В помощь лектору)

Ленинград
1989

ББК32.973
Б91

Бураго А. Ю., Кириллин В. А., Романовский И. В.
Б91 Форт — язык для микропроцессоров. — Л.: Знание. 1989. — 36 с.

(О-во «Знание» РСФСР. Ленингр. организация). 26 000 экз.

Рассказывается о недавно появившемся и бурно завоевывающем популярность языке программирования. Он служит простым и эффективным средством программирования для мини- и микро-ЭВМ. Его легко приспособить для конкретных областей применения. Язык Форт получил широкое распространение как язык для непосредственного управления оборудованием, в том числе роботами. Форт включен в состав школьного компьютера.

Брошюра предназначена для лекторов, а также для лиц, изучающих и интересующихся программированием на мини- и микро-ЭВМ, и для широкого круга читателей.

Б $\frac{2405000000-026}{073(02)-89}$ 28—88 **ББК32.973**

Издание рекомендовано секцией пропаганды естественнонаучных знаний при Правлении Ленинградской организации общества «Знание» РСФСР.

Рецензенты: С. Н. Баранов, кандидат физико-математических наук; С. М. Селеджи.

Эта брошюра является *начальным,* но *обстоятельным* пособием для изучения языка Форт (от англ. FORTH — вперед) и для самостоятельной работы на компьютере. Если Вы профессионал и знакомы с другими языками программирования, то на время забудьте их. Для работы с системой Форт необходимо знать лишь основы программирования.

Форт — необычный язык программирования. Основной его принцип — *в простоте.*

Форт — это язык одновременно высокого и низкого уровней, набор модулей-инструментов для создания программ и методология программирования.

Форт — это скорость, компактность, мобильность. Он реализован практически на всех отечественных и зарубежных компьютерах и особенно эффективен при разработке программ для микропроцессоров и микро-ЭВМ. Известно, что Форт сокращает время разработки программ благодаря модульности и диалогу.

Форт применяется:

при разработке трансляторов и операционных систем;

для управления станками, роботами, медицинским оборудованием;

в системах управления базами данных;

в задачах машинной графики;

в системах искусственного интеллекта.

Во всем этом есть одна тонкость. Форт дает разнообразные средства, но возлагает на программиста бо́льшую, чем в других языках, ответственность за эффективность их использования.

Программирование на Форте отличается прежде всего тем, что Форт — диалоговая система, вся работа на нем выполняется в присутствии человека, который должен дать ЭВМ задание и в процессе выполнения этого задания может или даже должен вмешиваться в вычисления.

Для общения Форт предоставляет человеку возможность ввода информации в ЭВМ и получения информации. Обычно в современных условиях для ввода и вывода используется дисплей — устройство, по-видимому, уже хорошо известное. Возможность «разговора с машиной» неопытного человека может испугать: если ясно, что машина может написать на экране дисплея вполне понятный текст, то другая часть общения («что же я скажу машине?») действительно загадочна.

Нужно знать, о чем Вы хотите поговорить с ЭВМ. В языке Форт имеется сравнительно небольшой словарь, с которым можно быстро освоиться и при его помощи дать машине задание.

В процессе диалога с ЭВМ Вы можете договариваться с ней о

введении новых слов, которые Вы ей сами объясняете с помощью уже известных. Это дает возможность каждому человеку формировать свой лексикон для общения с машиной, отражающий его собственные привычки, вкусы и вычислительные потребности. Можно переименовывать и первоначальные слова, например, если Вам не понравится слово SWAP (это английский глагол), его можно заменить русским словом ПЕРЕСТАВЬ, что будет хотя и понятнее, но длиннее.

Опыт показывает, что люди, работающие с Фортом, очень быстро привыкают к его первоначальному словарю и правилам формирования слов, так что такие замены остаются упражнениями для новичков.

Форт представляет большой интерес и тем, какой набор слов его авторы сочли необходимым включить в минимум. Первоначально многое может показаться необычным. Но не торопитесь делать критические выводы. Описываемые здесь средства были выбраны в результате долгой работы и внимательного анализа большим коллективом опытных программистов.

СЛОВА И ИХ ВЫПОЛНЕНИЕ

Система Форт предназначена для работы в диалоговом режиме. Программист, работающий с системой, имеет в своем распоряжении какое-либо техническое устройство, позволяющее ему вводить нужные тексты и получать ответы. Чаще всего в качестве такого устройства используется *видеотерминал* — своеобразное сочетание телевизора и пишущей машинки. На экране телевизора текст выглядит как на странице книги или машинописи (наверное, читатель уже видел персональные компьютеры или дисплеи крупных вычислительных машин или терминалы системы для продажи железнодорожных билетов). Текст появляется на экране по командам ЭВМ или, если инициатива передана работающему, при нажатии им клавишей клавиатуры.

При начальном запуске система представляется и передает управление программисту, ожидая от него ввода информации. Обычно в этом случае на экране появляется «подсказка» — условный значок, показывающий ожидание ввода. Программист должен набрать после «подсказки» необходимый текст, проверить и исправить его, если требуется, и передать системе нажатием клавиши ввода.

После этого система начинает исполнять приказы программиста, причем в этих приказах может быть предусмотрена печать текстов на экране. Действия, вызывающие появление текстов на экране дисплея, очень похожи на действия с пишущей машинкой, и мы будем называть такие действия печатью, хотя переноса информации на бумагу, так привычную для термина «печать», здесь обычно не бывает.

После того, как система выполнила все переданные ей приказы,

она переходит в состояние ожидания ввода. Программист набирает новый текст и снова нажимает клавишу ввода, тем самым продолжая диалог с Форт-системой.

Теперь о тексте, который программист передает системе. Основным понятием системы Форт является *слово*. Слово — это любая последовательность символов, отличных от пробелов.

Некоторые слова Форт-системе знакомы, другие — она узнает из вводимых текстов.

Текст — это любая последовательность слов, разделенных пробелами.

Вот пример текста:

Не забудь, пожалуйста , выключить электроприборы .

Этот текст состоит из семи слов, так как вторая запятая и точка в нашем определении слова — отдельные слова (очень важные для Форта).

АРИФМЕТИЧЕСКИЙ СТЕК

В языке Форт стеком называется хранилище *целых* чисел, используемое для размещения аргументов и результатов операций языка.

На каждое число в стеке отводится 2 байта (элемента памяти).

Хранимые в стеке числа упорядочены по положению. Будем считать, что последнее число находится *справа*. Позиция, занятая крайним правым числом, называется *вершиной* стека. Стек характеризуется тем, как выполняются основные операции хранилища — *добавление* и *удаление* информации. При добавлении числа в стек оно приписывается *справа* от имеющихся, при снятии числа со стека удаляется крайнее правое число.

Введенное число всегда помещается на вершину стека. Слово DROP (снять) снимает число с вершины стека. Слово . (точка) берет из стека число и печатает его. Слово S. печатает весь стек, оставляя его неизменным.

В дальнейшем действие слов, служащих для изменения состояния стека, будем показывать на диаграмме

стек *до* операции → стек *после* операции ,

причем не затрагиваемую операцией часть стека будем изображать многоточием.

Например, слово DUP (дублировать) добавляет в стек еще одно значение, равное тому, которое находится на вершине стека. Это слово представляется диаграммой

DUP (дублировать) ... a → ... a a

Далее без комментариев

DROP	(снять)	... a	→ ...
SWAP	(обменять)	... a b	→ ... b a
OVER	(через)	... a b	→ ... a b a
ROT	(вращать)	... a b c	→ ... b c a
—ROT		... a b c	→ ... c a b

2DROP	... a b	→ ...
2DUP	... a b	→ ... a b a b
2SWAP	... a b c d	→ ... c d a b
2OVER	... a b c d	→ ... a b c d a b
PICK (взять)	... a {n чисел} n	→ ... a {те же n чисел} a

(В частности, 0 PICK эквивалентно DUP, а 1 PICK — слову OVER).

ROLL (повернуть) ... a {n чисел} n → ... a {те же n чисел} a

(В частности, 1 ROLL эквивалентно SWAP, а 2 ROLL слову ROT). Слово DEPTH (глубина) кладет в стек число, равное глубине стека — количеству чисел, находившихся в стеке.

Проследим, например, выполнение следующего текста.

1 2 3 DUP DEPTH —ROT 2OVER S.

Выпишем текст в столбик, сопровождая каждое слово состоянием стека после его исполнения.

1	(1)
2	(1 2)
3	(1 2 3)
DUP	(1 2 3 3)
DEPTH	(1 2 3 3 4)
—ROT	(1 2 4 3 3)
2OVER	(1 2 4 3 3 2 4)

А теперь сами:

1 2 3 OVER 5 —ROT DROP 7 2SWAP ROT S.

Какие слова надо выполнить, чтобы изменить стек в соответствии со следующей диаграммой:

n1 n2 n3 n4 → n4 n3 n2 n1

АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ

Арифметический стек — основное поле для выполнения арифметических действий и хранения промежуточных результатов вычислений. Даже простейшие арифметические операции берут аргументы из стека и помещают результаты обратно в стек.

Знак операции (точнее слово, обозначающее операцию) пишется *после* того, как аргументы в стеке уже размещены. Текст

13 3 —

помещает в стек число 10, так как слово — (минус) извлекает из стека два числа, сперва вычитаемое, потом уменьшаемое, и помещает в стек их разность:

—	... a b	→ ... a—b

С другими операциями все обстоит аналогично:

+	... a b	→ ... a+b
×	... a b	→ ... a×b
ABS (абсолютное значение)	... a	→ ... |a|
NEGATE (обратный знак)	... a	→ ... —a

/	... a b → ...	целая часть
MOD	... a b → ...	остаток
/MOD	... a b → ...	остаток целая часть

В трех последних словах имеются в виду остаток и целая часть частного от деления a на b.

Таким образом, если в стеке лежат числа 26 и 7, то слово / превратит их в 3, слово MOD в 5, а слово /MOD в пару чисел 5 и 3.

Имеются специальные слова для действий с 1 и 2 (они выполняются немного быстрее):

1+ ... a → ... a+1

Аналогично работают 1— 2+ 2— 2× 2/ .

Числа на стеке могут восприниматься различным образом в зависимости от того, какое слово их использует. В отводимых двух байтах (16 двоичных разрядах) может уместиться лишь $2^{16}=65536$ различных значений числа. Обычно они трактуются как числа из диапазона от -2^{15} до $2^{15}-1$, но есть слова, которые воспринимают их как числа от 0 до $2^{16}-1$. Следующие слова выполняют поразрядные логические операции над двоичным представлением чисел; в этих операциях числа трактуются как наборы из шестнадцати битов:

AND (И)	... a b → ...	aANDb
OR (ИЛИ)	... a b → ...	aORb
NOT (НЕ)	... a → ...	NOTa
XOR (ИСКЛЮЧАЮЩЕЕ ИЛИ)	... a b → ...	aXORb

НОВЫЕ СЛОВА

Помимо уже существующих в системе слов можно создавать новые, определяя их действия через уже известные системе. Исполнение созданного слова — это исполнение действий, входящих в его *описание*. Описание начинается словом : (двоеточие) и кончается словом ; (точка с запятой). Сразу после слова : стоит описываемое слово, и далее — входящие в него действия. Например, текст:

: S2 DUP × SWAP DUP × + ;

определяет слово S2, вычисляющее сумму квадратов двух чисел из стека,

S2 ... a b → .. a×a+b×b

При разработке новых слов нужно внимательно следить за изменениями стека, делать комментарии. Комментарии ограничиваются скобками: слово ((открывающая скобка) пропускает следующий за ним текст до первого символа) (закрывающая скобка), например,

```
: S2            (    a b → a×a+b×b                            )
    DUP         (... a b b     — теперь  только  результат)
    × SWAP      (... b×b a     —    »       »        »    )
    DUP ×       (... b×b a×a —    »       »        »    )
    + ; (определение закончено)
```

После *выполнения описания* слова сразу же могут использоваться и в вычислениях и в определении других слов. Например, сумму четырех квадратов можно определить так:

```
: S4  (a b c d → a×a+b×b+c×c+d×d)
      S2 —ROT S2 + ;
```

Слово WORDS (слова) выводит на экран список всех уже известных системе слов (от самых новых к самым старым). Можно отменить уже определенное слово («забыть» его), но при этом забываются также и все слова, определенные позже. Для этого нужно выполнить текст из слова FORGET (забыть) и забываемого слова:

FORGET S2

Такой способ «забывания» вполне естественен: система располагает в памяти машины информацию о вновь определенных словах подряд, и забывание — это освобождение памяти тоже подряд, т. е. всей памяти, начиная с некоторого места. Можно сказать, что память, отведенная для определений слов, организована как стек.

Вы наверное обратили внимание на то, что при выполнении описания (например, описания слова S2) входящие в него уже известные слова не должны выполняться сразу же (слово DUP не дублирует вершину стека и т. д.), иначе не получится заготовки слова для последующего исполнения. В любой момент времени система находится в одном из двух состояний — *исполнения* или *компиляции*. Обычно система находится в первом состоянии, в котором вводимые слова сразу же исполняются. Исполнение слова : (двоеточие) переводит систему в состояние компиляции, при котором вводимые слова запоминаются для последующего исполнения.

Приведем еще два примера. Слово 8MOD эквивалентно тексту 8 MOD, но использует логические операции. Интересно слово LAST1 (последний бит), выделяющее в двоичном разложении числа младшую единицу.

```
: 8MOD 7 AND ;
: LAST1 DUP DUP 1— XOR AND ;
```

Это последнее слово — хорошее упражнение на применение логических операций.

Слово ; завершает компиляцию и переводит систему в режим исполнения. Таким образом, само это слово не компилируется, а исполняется, несмотря на режим компиляции. Это происходит потому, что оно обладает специальным признаком *немедленного исполнения*.

УСЛОВНОЕ ИСПОЛНЕНИЕ

При определении нового слова могут потребоваться знакомые Вам из других языков конструкции, организующие условное и циклическое исполнение. Для этого должны быть предусмотрены и логические величины, принимающие традиционные значения *истина* и *ложь*. Эти значения представлены целыми числами, причем *истина* соответствует числу —1 (двоичные разряды этого числа состоят из 16 единиц), а *ложь* соответствует 0 (16 двоичных нулей). Для них имеются и стандартные константы TRUE (истина) и FALSE (ложь), кладущие на стек соответственно числа —1 и 0. Логические значения получаются при выполнении некоторых слов, предназначенных для сравнения чисел.

Слова арифметического сравнения:

```
>   ... a b → ... a>b (т. е. при a>b TRUE
                       иначе FALSE)
<   ... a b → ... a<b
=   ... a b → ... a=b
0=  ... a   → ... a=0
0>  ... a   → ... a>0
0<  ... a   → ... a<0
```

Над логическими значениями можно совершать логические операции, приведенные ранее; заметим, что константы TRUE и FALSE подобраны так, что после выполнения логических операций над логическими значениями снова получаются логические значения. Описываемые дальше управляющие конструкции воспринимают числа из стека как логические значения следующим образом: 0 соответствует значению *ложь*, любое другое число — *истине*.

Для организации условного исполнения в языке Форт предусмотрены слова IF (если), ELSE (иначе) и THEN (то). Они используются в конструкциях

IF <часть-если> ELSE <часть-иначе> THEN

IF <часть-если> THEN.

Слово IF берет из стека логическое значение, и, в случае, если это значение — *истина*, т. е. не нуль, исполняет текст <часть-если>; в противном же случае исполняется <часть-иначе>, если она есть. Дальше управление передается на текст, следующий за THEN. Заметим, что использование управляющих слов требует состояния компиляции; таким образом, можно использовать их только при определении новых слов.

Пример. Стандартное слово ABS можно было бы определить так:

```
: ABS        ( ... a → ... |a| )
    DUP 0 <  ( a    a<0   )
    IF       ( a          )
    NEGATE THEN ;
```

Пример другой конструкции разберите сами и постарайтесь улучшить.

```
: MAX ( ... a b → max {a, b} )
     2DUP > IF DROP ELSE SWAP DROP THEN ;
```

Слово MAX и аналогичное ему MIN входят в стандарт языка Форт.

Конечно же, конструкции IF могут быть вложенными; нужно только следить за скобочными соответствиями слов IF и THEN.

ЦИКЛЫ

Для организации циклов в языке Форт предусмотрены слова BEGIN (начало), WHILE (пока), REPEAT (повторить) и UNTIL (пока — не), используемые в конструкциях

BEGIN <часть-начало> WHILE <часть-повторение> REPEAT
и
BEGIN <часть-начало> UNTIL.

В последней конструкции после исполнения текста <часть-начало> слово UNTIL берет из стека оставленное этим текстом логическое значение; в том случае, если это значение *ложь*, снова исполняется <часть-начало>, потом UNTIL и т. д.; итерации прекращаются, когда UNTIL возьмет из стека значение *истина*. Пример вычисления факториала может выглядеть так:

```
: FACT                  ( ... n → ... n!              )
  DUP 2< IF DROP 1      ( 1 если n<2, то n!=1         )
  ELSE                  ( n иначе                     )
  DUP                   ( n n s=n  k=n                )

      (Теперь лежащие в стеке числа будут представлять)
      (s — накопленное произведение  и  k — множитель )

BEGIN                   ( s k                         )
1—                      ( s k'    k'=k—1              )
SWAP OVER × SWAP        ( s' k'   s'=s×k              )
DUP 1 =                 ( s k  k=1 если k=1, то s=n!  )
UNTIL                   ( n! 1 иначе повторить        )
DROP THEN ;             ( n!                          )
```

Конструкция цикла типа WHILE

BEGIN <часть-начало> WHILE <часть-повторение> REPEAT используется, когда в цикле есть действия, которые в заключительной итерации не нужны: первоначально выполняется <часть-начало>, слово WHILE снимает со стека логическое значение и, если это *истина*, то выполняются тексты <часть-повторение>, <часть-начало>, снова слово WHILE и т. д. Когда слово WHILE снимет со стека *ложь*, выполнение цикла закончится и начнет выполняться текст, следующий после REPEAT.

Отметьте, что значение *истина* здесь соответствует *продолжению* вычислений, в отличие от циклов типа UNTIL.

Пример 1. Наибольший общий делитель двух положительных чисел.

```
: НОД  ( a b → НОД [ a, b ]    по Евклиду)
2DUP < IF SWAP THEN          ( теперь a > = b           )
BEGIN DUP                    ( a b b                    )
WHILE                        ( пока b > 0               )
2DUP MOD                     ( a b      aMODb           )
ROT                          ( b aMODb a                )
DROP                         ( a′ b′  — новые значения a и b )
REPEAT DROP ;                ( НОД  (a, b)     (все)    )
```

Пример 2. Подсчет числа единиц в двоичном разложении числа.

```
: UNITS  ( a → число единиц в a )
0 SWAP                       ( 0 a                      )
   ( В стеке лежат два числа: счетчик числа единиц s )
   и постепенно изменяемое число a                   )
BEGIN                        ( s a′                     )
DUP                          ( s a′ a′                  )
LAST1 DUP                    ( s a′ d d                 )
WHILE                        ( пока d > 0               )
  —                          ( s a″                     )
SWAP 1+SWAP                  ( s′ a″                    )
REPEAT 2DROP ;               ( s                        )
```

Более сложная конструкция цикла — *цикл со счетчиком* — описана дальше.

КОНСТАНТЫ И ПЕРЕМЕННЫЕ

Вы уже познакомились с одним из способов введения новых слов при помощи определяющего слова : . Сейчас появятся еще два определяющих слова:

Слово CONSTANT (константа) в тексте

CONSTANT <имя>

определяет новое слово <имя> как константу со значением, равным числу на вершине стека, и снимает со стека это число. В дальнейшем выполнение слова <имя> кладет это число в стек. Так, после исполнения текста

80 CONSTANT LINESIZE

слово LINESIZE (длина строки) будет класть в стек число 80.

Определяющее слово VARIABLE (переменная), которое используется в конструкции:

VARIABLE <имя>

резервирует в памяти компьютера 2 байта под значение переменной <имя>. Исполнение слова <имя> кладет в стек число — адрес зарезервированного места. Этот адрес может использоваться другими словами.

Вот три важных слова, использующих адреса:

Слово ! (восклицательный знак, читается «запомнить») служит для записи числа по данному адресу (addr):

! ... a addr → ... [addr] : =a

Слово @ (читается «взять») кладет в стек число, лежащее по адресу, взятому из стека:

@ ... addr → ... a a=[addr]

Слово +! прибавляет величину a к числу по адресу addr:

+! ... a addr → ... [addr] : = [addr]+a

Пример. Текст

A @ 5 + B ! (A и B — переменные)

соответствует оператору B : =A+5 других языков программирования.

Слова, определенные с помощью CONSTANT и VARIABLE, практически ничем не отличаются от других слов. В частности, их можно «забывать» с помощью слова FORGET. Сами слова CONSTANT, VARIABLE и : являются частными случаями более общих конструкций — определяющих слов. Со способами их задания читатель познакомится ниже.

КОДОФАЙЛ

Основной участок памяти, находящийся в распоряжении программиста, называется *кодофайл*. В нем, в частности, располагается и словарь системы, включая память, зарезервированную под константы и переменные. Память распределяется по возрастанию адресов — свободная память находится в конце кодофайла. Таким образом, кодофайл тоже представляет собой стек — стек памяти. Мы будем называть *вершиной кодофайла* первый свободный байт памяти. Как видите, в кодофайле размещены как программа, так и информационные объекты. Такое размещение требует от программиста особой осторожности в работе с памятью — изменения в ячейках памяти с неправильным адресом нарушают работу системы.

Вот некоторые базовые слова для работы с кодофайлом.

На стек кладется адрес вершины кодофайла:

HERE (здесь) ... → ... addr

Слово ALLOT резервирует n байтов свободной памяти — адрес вершины кодофайла увеличивается на n (а при n<0 уменьшается):

ALLOT (занять) ... n → ...

Занятие двух байтов в кодофайле и запись туда n:

```
,            ... n → ...
```

При работе с памятью кодофайла широко используются слова @ и ! .

Пример. Следующее слово резервирует в кодофайле память под n целых чисел (число n берется из стека) и кладет в стек адрес начала зарезервированного места:

```
: 2ALLOT                ( n → a   )
        HERE                ( n a )
        SWAP                ( a n )
        2 × ALLOT ;
```

Теперь легко описать, например, слова, заменяющие привычную по другим алгоритмическим языкам конструкцию массива:

```
: (I)                   [ i a → a (i) ]
        OVER + + ;
```

После выполнения следующего текста

```
N                 ( N — константа — число элементов массива)
2ALLOT            (на стеке адрес начала массива          )
CONSTANT B
```

текст B (I) будет снимать со стека номер элемента в массиве B и класть в стек адрес этого элемента.

```
B (I)             ( i → addr )
```

Разумеется, никаких проверок корректности номера элемента не делается.

СИМВОЛЫ

Для представления символьной информации отводится по одному байту памяти на каждый символ. Таким образом, каждому символу сопоставляется число от 0 до 255, которое называется его *кодом*. В разных ЭВМ используются разные *кодировки*. В отечественных машинах, где требуется добавить к множеству символов кириллицу, наиболее распространена кодировка КОИ-8, являющаяся расширением распространенной в мире кодировки ASCII. Кодировка КОИ-8 приведена в приложении К.

Имеются слова для работы с отдельными символами:

```
C@                ...  addr → ... c
```

В стек помещается число, равное содержимому байта по адресу addr. (Отметьте, что адрес обычного двухбайтового целого числа — это адрес его правого байта).

```
C !               ... c addr → ...
```

В байт по адресу addr записывается символ c.

C, ... c → ...

Слово, аналогичное слову , (запятая), но резервирующее (и записывающее) только один байт.

KEY (клавиша) ... → ... c (ожидание)

При выполнении этого слова система переходит в ожидание, пока не будет нажата клавиша какой-либо литеры на клавиатуре дисплея. Код этой литеры и кладется в стек.

EMIT (выдать) ... c → ...

Символ c будет напечатан.

Слово BL кладет в стек код пробела. Во многих реализациях встречается очень полезное слово C", которое кладет в стек код первой следующей за ним литеры, не являющейся пробелом. Слово C" делает текст более наглядным, чем при непосредственном использовании кодов. Например, чтобы напечатать знак плюс, нужно выполнить текст

C" + EMIT

А как быть с кодом пробела? Его кладет в стек слово BL.

РАБОТА С УЧАСТКАМИ ПАМЯТИ

Часто приходится выполнять действия сразу над большими участками памяти. Участок памяти в таких действиях определяется адресом его начального байта и длиной.

FILL (запомнить) ... addr n c → ...

Содержимое n байтов, начиная с адреса addr, заполняется младшим байтом c.

BLANK (заполнить пробелами) ... addr n → ...

Эквивалентно FILL с заполнением кодом пробела.

ERASE (стереть) ... addr n → ...

Эквивалентно тексту 0 FILL.

CMOVE (переслать) ... addr1 addr2 n → ...

Перемещение участка в n байтов с началом addr1 по адресу addr2.

Слово CMOVE> отличается от CMOVE тем, что начинает перемещение с *последнего* байта участка. Различие этих слов существенно при перекрытии участков. Вот небольшой пример. Пусть на вершине стека лежит адрес участка памяти в 12 байтов, где записан текст «Форт-система». Тогда исполнение

DUP 4 + 8 CMOVE

превратит этот текст в «ФортФортФорт», а если выполнить

CMOVE> вместо CMOVE, то получится «ФортФорт-сис». Заметим, что с помощью этих слов можно выполнить иногда очень важные действия — размножить маленькие участки памяти внутри больших (например, числа внутри массивов) подобно тому, как в нашем примере размножилось слово «Форт».

Для заполнения участка памяти информацией, вводимой непосредственно с клавиатуры, имеется слово EXPECT (ожидать)

EXPECT ... addr n → ...

Участок памяти от адреса addr длиной n байтов заполняется от начала к концу вводимыми с клавиатуры символами до тех пор, пока не заполнится весь участок или программист не завершит ввод (нажав клавишу ввода). Переменная SPAN содержит число введенных символов. Результат ввода автоматически выводится «на печать».

СТРОКИ

Строкой со счетчиком (в дальнейшем просто *строкой*) называется участок памяти, в первом байте которого. находится *счетчик* — **байт, хранящий длину участка (без учета байта длины).** Адресом строки считается адрес счетчика.

Слово " «кавычка» употребляется в конструкции

"<текст>" ... → ... addr

Текст <текст> будет превращен в строку, расположенную во временном буфере, адрес этой строки кладется в стек. Слово COUNT преобразует адрес строки в адрес и длину ее текста:

COUNT ... addr → ... addr + 1 n ,

а слово TYPE выводит текст (участок памяти) по его адресу и длине:

TYPE (напечатать) ... addr n → ...

В качестве примеров работы со строками рассмотрим тексты:

" МОЛОДЕЦ " COUNT TYPE (напечатается МОЛОДЕЦ)

" МОЛОДЕЦ " COUNT 3 — TYPE (напечатается МОЛО)

Разберите следующие примеры: слово S, размещает в кодофайле строку с данным адресом, а слово T, — ее текст. Оба слова оставляют на стеке адрес получившегося объекта:

```
: T,            ( a1 — адрес строки → a2 — адрес в кодофайле )
  HERE SWAP          ( a2 a1                         )
  COUNT              ( a2 a1+1 n                     )
  HERE OVER ALLOT    ( a2 a1+1 n a2                  )
  SWAP CMOVE ;       ( a2                            )
: S,                 ( a1 — адрес строки → a2 — адрес
                       строки в кодофайле            )
```

```
HERE SWAP
   DUP C@ 1+          ( a2 a1 n+1                          )
HERE OVER ALLOT       ( a2 a1 n+1 a2 — отведено n+1 байт )
SWAP CMOVE ;
```

Слово . ″ употребляется в конструкции

. ″ <текст> ″ ,

при выполнении которой текст <текст> будет выведен на терминал.

Упомянем еще несколько возможностей вывода: слово CR переводит строки, а слово SPACE вставляет в выходной текст пробел (т. е. оно эквивалентно BL EMIT или . ″ ″).

ЦИКЛЫ СО СЧЕТЧИКОМ

Выше рассматривались циклы типа WHILE и UNTIL. Всем известно, что не менее важны, в частности, для работы с участками памяти, *перечислительные циклы*, в которых используется *параметр цикла* — целочисленная переменная, пробегающая нужное множество значений. Для организации таких циклов используется (внутри определений слов) конструкция:

```
DO <тело-цикла>  +LOOP
 или
DO <тело-цикла>    LOOP
```

Слово DO (делать) берет из стека два значения:

DO ... b a →,

где a — начальное значение параметра цикла, a b — пороговое. Слово +LOOP (цикл) прибавляет в каждой итерации к параметру цикла число, которое берет из стека. Текст <тело-цикла> исполняется до тех пор, пока очередное значение параметра цикла не «перепрыгнет» через границу, проходящую между b—1 и b. Конструкция

DO ... +LOOP

допускает и отрицательные приращения параметра цикла; именно с учетом этого и дана такая хитрая формулировка правила завершения цикла. Используемое чаще слово LOOP эквивалентно 1 +LOOP. В конструкции с этим словом <тело-цикла> исполнится b—a раз при b>a. При b⩽a цикл DO ... LOOP выполняется 65536 + b—a раз.

Слово I помещает в стек текущее значение параметра цикла. Естественно, что циклы могут быть вложенными; значение параметра объемлющего цикла помещается в стек словом J.

Пример. Сумма квадратов первых n натуральных чисел:

```
: SS2  ( n → сумма )
       0 SWAP 1+ 1 DO I DUP × + LOOP ;
```

Слово LEAVE обеспечивает немедленный выход из цикла (самого глубокого).

Возникает естественный вопрос, куда же деваются значения a и b, которые DO снимает со стека, и откуда берется текущее значение параметра цикла.

В системе есть еще один стек — *стек возвратов*, о нем будет рассказано ниже.

СЛОВАРНАЯ СТАТЬЯ

Теперь более подробно рассмотрим, как функционирует Форт-система. Она может быть устроена по-разному, но наиболее распространены два способа организации — с помощью прямого и косвенного шитых кодов. Там, где это возможно, мы будем описывать устройство и работу систем независимо от способа реализации; в остальных же случаях будем придерживаться реализации с помощью *прямого шитого* кода.

Для каждого слова при его определении система создает *словарную статью* — расположенный в кодофайле информационный объект, содержащий перечень входящих в слово действий и устроенный так, чтобы был возможен его поиск по имени слова в словаре системы.

Можно считать, что словарная статья состоит из *системной* части, служащей для ее хранения и поиска, и *программной* части, описывающей действия и информацию, связанные с этим словом.

В *системной* части естественно выделяются:

1. *Поле имени,* содержащее имя слова, представленное в виде строки со счетчиком.

2. *Поле связи* — адрес словарной статьи предыдущего слова, служащий для организации списка словарных статей.

В *программную* часть включаются:

1. *Поле кода,* или *исполняемая часть,* представляющая собой *вызов интерпретатора* (для прямого кода это — машинная команда, передающая управление специальной программе — *адресному интерпретатору*).

2. *Поле параметра,* или *данные* — область памяти, используемая словом (например, зарезервированная память в слове, определенном с помощью VARIABLE).

При выполнении введенного слова его прежде всего ищут в словаре; если слово найдено, то оно исполняется, т. е. управление передается на поле кода этого слова. Для выполнения каждого из этих двух действий по отдельности служат следующие слова:

слово ′ (читается штрих) употребляется в конструкции

′ <слово> ... → ... addr

Слово <слово> должно быть уже определено. При выполнении этой конструкции адрес исполняемой части слова <слово> кладется на стек.

Слово EXECUTE (выполнить)

EXECUTE ... addr → ...

снимает со стека адрес исполняемой части некоторого слова и выполняет это слово. Таким образом, текст

' <слово> EXECUTE

эквивалентен тексту

<слово>

Заметим, что при определении нового слова входящие в него слова не выполняются, а происходит вот что: адреса их исполняемых частей заносятся в кодофайл, формируя поле параметров определяемого слова.

КОМПИЛЯЦИЯ И ИСПОЛНЕНИЕ

В каждый момент система может находиться в одном из двух состояний — *исполнения* и *компиляции*. В состоянии исполнения каждое введенное слово разыскивается в словаре и *исполняется;* в состоянии же компиляции оно не исполняется, а *компилируется,* т. е. адрес его исполняемой части заносится на вершину кодофайла с резервированием этого места. При выполнении слова : (двоеточие) система переходит из режима исполнения в режим компиляции, а в кодофайл вписывается системная часть определяемого слова и вызов интерпретатора. Далее слова, входящие в определение нового слова, компилируются, формируя его поле кода.

Слово ; завершает компиляцию и переводит систему в режим исполнения. Таким образом, само это слово не компилируется, а *исполняется*, несмотря на режим компиляции. Вы уже знаете, что оно обладает специальным признаком *немедленного исполнения.* Тем же признаком обладают и все управляющие конструкции (условного и циклического исполнения); конечно, среди выполняемых ими действий есть и компиляция некоторых специальных слов.

Предусмотренное в системе слово IMMEDIATE (немедленное) присваивает признак немедленного исполнения последнему определенному слову.

Для управления режимами исполнения и компиляции предусмотрены следующие слова:

[— перевод системы в режим исполнения; естественно, слово [обладает признаком немедленного исполнения.

] — перевод системы в режим компиляции.

Слово [COMPILE] (компилировать) выполняет принудительную компиляцию следующего за ним слова независимо от наличия у него признака немедленного исполнения. Само оно также имеет признак немедленного исполнения.

В выполняемые словом действия может входить компиляция других слов. Такая «компиляция компиляции» описывается конструкцией

COMPILE <слово>

Заметим, что слово, включающее такие действия, имеет обычно признак немедленного исполнения.

В системе имеется удобное слово FIND (искать), служащее для поиска слов с проверкой признака немедленного исполнения:

FIND ... addr1 → addr2 n

Здесь addr1 — адрес строки со счетчиком, содержащей имя слова. Число n принимает значение 0, если слово не найдено, 1, если слово найдено и имеет признак немедленного исполнения, —1, если этого признака нет. Значение addr2 в первом случае равно addr1, в остальных — задает адрес поля кода.

В отличие от ' слово FIND использует строку со счетчиком, это позволяет формировать образец поиска программным путем.

Поговорим теперь немного об управляющих конструкциях. Они используют стандартные слова BRANCH (переход) и ?BRANCH. Скомпилированное слово BRANCH при своем исполнении изменяет текущий адрес интерпретации на адрес, записанный сразу после него, выполняя тем самым безусловный переход, так как после этого будет выполняться программный текст, расположенный по этому адресу; слово ?BRANCH выполняет условный переход: оно снимает со стека число, и переход происходит лишь в том случае, если это число — 0.

Для примера посмотрите, как можно реализовать слова IF и THEN:

```
: IF COMPILE ?BRANCH HERE 0 , 2 ; IMMEDIATE
: THEN 2  ?PAIRS HERE  SWAP  !  ;  IMMEDIATE
```

Слово ?PAIRS снимает со стека два числа и выдает сообщение об ошибке, если они не равны; это слово используется для проверки правильности записи управляющих конструкций; в данном примере номер 2 зарезервирован для условных операторов.

Таким образом, исполнение слова IF компилирует адрес слова ?BRANCH, резервирует место для ссылки вперед (на обход ветви — IF) и оставляет адрес зарезервированного места на стеке. Слово THEN снимает этот адрес со стека и вписывает в него текущий адрес в кодофайле.

Заметьте, что стек, в режиме компиляции программистом не используемый, активно используется в процессе компиляции слов системой, поэтому изменять его во время исполнения определений (например, текстом [DROP]) не рекомендуется.

СТЕК ВОЗВРАТОВ

Для того чтобы слегка отдохнуть от трудного материала предыдущих разделов, познакомимся с более простой конструкцией — стеком возвратов, знание которой очень пригодится при дальнейшем изложении.

Кроме арифметического стека, в системе используется еще один

стек, называемый *стеком возвратов*. Система использует его для организации циклов и вложенного исполнения слов.

Программист может временно хранить информацию в стеке возвратов с помощью слов:

>R   ... a → ... | ...   → ... a

число a снимается из стека и кладется в стек возвратов (справа от черты);

R>   ...   → ... a | ... a → ...

число a снимается из стека возвратов и кладется в арифметический стек;

R@   ...   → ... a | ... a → ... a

число a с вершины стека возвратов копируется в арифметический стек.

Пример. Описание слова 3DUP:

```
: 3DUP ( a b c → a b c a b c )
      >R 2DUP R@ —ROT R> ;
```

Ограничения: содержимое стека возвратов следует восстанавливать в прежнем виде при завершении исполнения слова и тела цикла со счетчиком. При использовании стека возвратов внутри цикла слова I и J могут выдавать неправильные значения: на стеке возвратов хранятся текущие и граничные значения параметров цикла. Слово I просто снимает нужное значение со стека возвратов в соответствии с выбранным в реализации форматом. Именно поэтому опасно совместное использование слов I и J с манипуляцией со стеком возвратов.

ИСПОЛНЕНИЕ СЛОВ

Теперь подробно опишем процедуру исполнения слов. Подчеркнем, что предлагаемое описание относится к *прямому шитому* коду; для косвенного кода эта процедура несколько иная.

Как уже говорилось, в системе имеется специальная программа — *адресный интерпретатор*, которая занимается исполнением слов, не записанных в машинных командах. Интерпретатор представляет собой программу с тремя режимами.

Исполнение слова начинается с вызова режима CALL (вызов), который устанавливает указатель интерпретации IP на начало кода данного слова. Прежнее значение IP, в котором хранилась точка вызова исполняемого слова, запоминается в стеке возвратов, что обеспечивает вложенное исполнение слов. Режим CALL завершается первым вызовом режима NEXT (следующий).

Режим NEXT передает управление по адресу, записанному в месте памяти, на которое указывает IP, одновременно передвигая IP на следующий элемент кода (т. е. прибавляя к нему 2).

Такие вложенные вызовы продолжаются до тех пор, пока управ-

ление не передано на машинную подпрограмму, которая и будет исполнена. В конце каждой такой программы записан вызов режима NEXT адресного интерпретатора. В конце каждого интерпретируемого кода записан вызов третьего режима интерпретатора — режима RETURN (возврат), его компилирует туда слово ; . Режим RETURN завершает исполнение слова. Он загружает IP значением, снимаемым со стека возвратов и вызывает режим NEXT.

Еще несколько замечаний к описанной процедуре:

1. При компиляции чисел перед числом компилируется специальное слово, кладущее на стек число из следующих за ним двух байтов и одновременно переставляющее указатель интерпретации после этих двух байтов. Аналогичным образом обрабатываются строки.

2. Слово EXIT позволяет закончить обработку данного слова в любом его месте; понятно, что это слово осуществляет вызов режима RETURN.

ОПРЕДЕЛЯЮЩИЕ СЛОВА

Вы уже знакомы с определяющими словами, служащими для образования новых слов. Это были слова CONSTANT и VARIABLE. Сейчас вы увидите способ описания таких слов; в дальнейшем будет удобно называть их также *словами-генераторами.*

Слова-генераторы описываются с помощью слов CREATE (создать) и DOES> (исполнить) в конструкции:

: <имя-генератора> CREATE <создающая часть>
DOES> <исполняющая часть> ;

и употребляются в конструкции

<имя-генератора> <имя>

для определения слова <имя> (сравните с использованием генераторов CONSTANT и VARIABLE).

При выполнении конструкции слово CREATE создает в кодофайле *словарную статью* для слова <имя>, а тем самым заносит слово <имя> в словарь системы, после чего исполняется текст <создающая часть>, который может зарезервировать в кодофайле дополнительное место и заполнить его по своему усмотрению, создав этим *поле параметров* слова. В дальнейшем при исполнении слова <имя> на стек кладется адрес поля параметров и выполняется текст <исполняющая часть>. Например,

```
: CONSTANT CREATE ,  ( слово , резервирует 2 байта      )
                     ( и кладет в них число из стека     )
  DOES>              ( на стеке адрес этих двух байтов   )
    @ ;              ( значение помещается в стек        )
```

Таким образом, <исполняющая часть> слова-генератора является интерпретатором для поля параметров определяемого слова, в поле кода которого вызов интерпретатора как раз и содержит ссылку на нее.

В качестве упражнения разберите устройство генератора одномерных массивов. Слово ARRAY (массив) берет из стека целое число S и резервирует в кодофайле место для S чисел, связывая с ними следующее за ARRAY слово как имя этого массива. В дальнейшем выполнение имени массива берет из стека индекс, проверяет его принадлежность к диапазону от 1 до S и, если индекс принадлежит этому диапазону, помещает в стек адрес соответствующего элемента массива:

```
: ARRAY                  ( в стеке лежит число элементов   )
   CREATE DUP ,          ( это число помещается в поле па-
                                               раметров )
   2× ALLOT              ( захват места для массива        )
   DOES>                 ( при вызове в стеке лежит индекс )
  OVER 1 <               ( и помещается адрес захваченной
                                                 памяти )

   IF    . " ИНДЕКС МЕНЬШЕ 1 " 2DROP
   ELSE 2DUP @ > IF . "ИНДЕКС БОЛЬШЕ ЧЕМ НАДО"
                   2DROP
             ELSE 1 + SWAP 2× + THEN THEN ;
```

АРИФМЕТИКА ДВОЙНОЙ ТОЧНОСТИ

Наряду с обычным представлением целых чисел, в котором на каждое число отводится по два байта, Форт допускает представление чисел с двойной точностью — число размещается в четырех байтах и, следовательно, может принимать значения от -2^{31} до $2^{31}-1$.

Чтобы отличать числа двойной точности от обычных чисел, надо в запись таких чисел (в любом месте) включать точку. Например 12345678 — число двойной точности.

Для размещения такого числа в стеке и, аналогично, в памяти отводятся две стандартные позиции, идущие подряд, левая половина числа располагается на вершине стека, а правая — за ней. Слов, работающих с числами удвоенной точности, в языке не очень много; разумеется, к ним относятся и давно известные нам слова

2DROP 2DUP 2OVER 2SWAP

Слово 2@ аналогично слову @, оно кладет на стек число двойной длины, находящееся по данному адресу:

2@ ... addr → ... d (двойное число)

Слово 2! аналогично слову ! :

2 ! ... d addr → ...

Из арифметических операций выбран самый минимальный набор

```
D+        ... d1 d2 → ... d1+d2
D—        ... d1 d2 → ... d1—d2
DABS      ... d     → ... |d|
DNEGATE   ... d     → ... —d
D<        ... d1 d2 → ... d1<d2
D=        ... d1 d2 → ... d1=d2
```

Операции умножения и деления являются «смешанными», умножение двум числам обычной длины сопоставляет их произведение двойной длины, в делении делимое имеет двойную длину, а делитель, частное и остаток — обычную:

```
M×        ... n1 n2 → ... d=n1×n2
M/MOD     ... d  n  → ... n1 n2 (остаток частное)
```

Для перевода числа в двойное используется слово S>D:

```
: S>D ( n → d )
  DUP 0< ;
```

Обратный перевод требует тщательной проверки его корректности, но в простейшем случае эквивалентен DROP.

Наконец, имеются слова 2CONSTANT и 2VARIABLE, вполне аналогичные своим прообразам для чисел обычной длины.

ФОРМАТНЫЙ ВЫВОД

До сих пор рассматривались только самые простые способы вывода информации. Можно к ним добавить еще по материалам предыдущего раздела слово D . , выводящее на печать двойное слово, снимаемое со стека. В основе этого и других слов, управляющих выводом, лежит несколько вспомогательных слов, с помощью которых можно организовать любой вывод.

В Форте можно просто менять систему счисления, используемую при вводе и выводе информации. Имеется переменная BASE (основание), хранящая основание текущей системы счисления. Ее исходное значение равно десяти (т. е. основанию обычной десятичной системы), и, следовательно, первоначально числа вводятся и выводятся в наиболее распространенной форме. При изменении значения BASE система счисления автоматически изменяется. Особенно часто используются шестнадцатиричная и двоичная системы, первая из них устанавливается словом HEX (шестнадцатиричная):

```
: HEX 16 BASE ! ;
```

Для возврата в десятичную систему используется слово DECIMAL (десятичная), которое без труда напишет читатель самостоятельно.

Описываемые ниже слова работают с *буфером вывода*, в котором формируется внешнее представление числа в виде строки сим-

волов. Форматное преобразование начинается словом <# , которое устанавливает указатель на конец буфера, так как формирование буфера идет от конца. Слово HOLD (сохранить) переносит литеру со стека в буфер и продвигает указатель буфера вперед на одну позицию. Слово #> завершает преобразование и кладет на стек адрес сформированной в буфере строки литер и ее длину.

Собственно для формирования символов должны использоваться другие слова, которые выделяют поочередно разряды числа и превращают их в соответствующие символы. Основным таким преобразователем является слово # . Оно делит двойное число с вершины стека на основание текущей системы счисления, заменяет его на стеке получившимся частным (тоже двойной длины), а остаток переводит в литеру и записывает в буфер при помощи слова HOLD.

Полный перевод числа выполняет слово # S, которое оставляет на стеке двойной нуль — результат последнего деления:

```
: #S ( d → 0 0 )
    BEGIN # 2DUP 0 0 D= UNTIL ;
```

Для вывода знака минус предусмотрено слово SIGN (знак):

```
: SIGN ( n → )
    0< IF C" — HOLD THEN ;
```

Итак, все готово для определения слова D. ;

```
: D. ( d → )
        2DUP DABS
    <# #S ROT SIGN #>
    TYPE SPACE DROP ;
```

Слово . для печати обычных чисел определяется совсем просто:

```
: .    ( n → )
           S>D  D. ;
```

В качестве примера форматного вывода создадим два полезных слова. Первое печатает номер телефона в стандартном виде:

```
: .PHONE ( d → )
    <# # # C" — HOLD
        # # C" — HOLD #S  #> TYPE ;
```

Проверим правильность этого слова:

```
2579469    .PHONE
257-94-69
```

При помощи второго слова можно выводить результаты марафонского забега, замеренного с точностью до сотых долей секунды, например:

2ч37м42.93с

Введем два вспомогательных слова. Слово SIXI устанавливает шестиричную систему счисления. Слово # MS выдает минуты или секунды:

```
: SIXI     (   →      )
                  6    BASE  !  ;
: # MS   ( d → d/60 )
               #  SIXI  #  DECIMAL ;
```

Слово .TABLEAU собственно и выводит результаты забега:

```
: .TABLEAU   ( d → )
   <# C" с  HOLD  #  #
         C" .  HOLD  #MS
         C" м  HOLD  #MS
         C"  ч  HOLD  #S  #>  TYPE ;
```

ВНЕШНЯЯ ПАМЯТЬ

Памяти микрокомпьютера обычно недостаточно для запоминания всех необходимых данных и программ. Для этого используется память на внешних носителях. В основном это память на магнитном диске, но можно воспользоваться и кассетным магнитофоном.

В Форте данные и программы запоминаются на диске *блоками* по 1024 байта (т. е. 1 Кбайт) каждый. Блоки иногда называют *экранами*. Имеется в виду, что блоки могут содержать текст, который распечатывается на экране в виде 16 строк по 64 символа в каждой.

Сначала рассмотрим, как можно распечатывать и загружать исходные тексты Форт-программ. Следующий оператор

25 LIST (распечатать)

распечатает на экране блок 25 в виде последовательности из 16 строк по 64 символа в каждой. Переменная SCR (номер экрана) хранит номер последнего распечатанного по LIST блока. Используя этот факт, можно написать следующие слова: L — распечатать блок еще раз; LL — распечатать предыдущий блок; LN — распечатать следующий блок:

```
: L    SCR @        LIST ;
: LL   SCR @   1— LIST ;
: LN   SCR @   1+ LIST ;
```

Слово INDEX (индекс) служит для распечатывания первых строк последовательности экранов. Обычно в первой строке экрана пишется комментарий. Например,

12 24 INDEX

распечатает первые строки экранов с 12 по 24 включительно.

Загрузка блока также проста, как его распечатка. Если блок содержит текст Форт-программы, которую вы с помощью редактора набрали с терминала, то этот текст может быть введен прямо из блока. Если вы напишите

25 LOAD

то текст из блока 25 будет обработан Форт-системой, как если бы он был введен с клавиатуры. Заметим, что блок 25 в свою очередь может содержать слово LOAD (загрузить), например,

45 LOAD

После загрузки блока 45 произойдет возврат и «дозагрузка» остатка блока 25.

Программа обычно занимает не один блок. Для загрузки последовательности блоков используется слово THRU (сквозь). Например,

12 24 THRU

загрузит блоки с 12 по 24 включительно.

Слово —> вызывает загрузку следующего блока. Слово ;S прекращает загрузку блока, с его помощью можно загрузить только часть блока.

Переменная BLK (номер блока) содержит номер блока, который загружается в текущий момент. Если BLK содержит ноль, это означает, что ввод текста идет с клавиатуры. Таким образом, блок с номером 0 не может содержать программу. Вторая переменная, используемая при загрузке блока, — это >IN (указатель ввода). Она содержит смещение в байтах относительно начала блока, откуда в настоящий момент идет ввод текста. Если BLK равно нулю, то >IN указывает на смещение от начала буфера ввода.

Существуют средства для работы с содержимым блока. Главным из них является слово BLOCK (блок), которое переносит блок с указанным номером на 1024-байтный *блочный буфер* в оперативной памяти и оставляет адрес начала этого буфера на стеке. Для работы с этим блоком теперь можно использовать любые слова, работающие с оперативной памятью. Если набрать

25 BLOCK 1024 TYPE ,

то распечатается текст, хранящийся в блоке с номером 25. Слова LIST и LOAD используют слово BLOCK для доступа к нужному блоку. Теперь попробуйте набрать

10 25 BLOCK !

Таким способом засылается в первую ячейку блока с номером 25 значение 10. Но это не значит, что содержимое блока на диске изменилось. Для такого изменения необходимо переслать содержимое блока 25 обратно на диск. Слово UPDATE (изменить) создано специально для этой цели. UPDATE помечает буфер, в котором хранится блок как измененный. И теперь, когда этот буфер потребуется для размещения какого-нибудь другого блока, он будет предварительно записан на диск, заменяя прежнее значение блока.

Можно заставить систему записать измененный блок на диск, не дожидаясь пока его вытеснит другой блок. Для этого используются слова FLUSH (очистить) и SAVE-BUFFERS (сохранить

буферы). SAVE-BUFFERS выгружает все помеченные по UPDATE блоки на диск. FLUSH после этого еще и очищает буферы, заполняя их пробелами.

Если же Вы случайно испортили буферы и не хотите, чтобы они были записаны на диск, то используйте в этом случае слово EMPTY-BUFFERS (очистить буферы). Оно сотрет все признаки UPDATE и очистит буферы, отменив тем самым все последние исправления. Восстановить буферы можно, снова загрузив их с диска при помощи слова BLOCK.

Описанных средств вполне достаточно, чтобы на их основе развивать файловые системы или организовывать базы данных. Данные в таких системах обычно хранятся в последовательно идущих блоках, доступ к которым осуществляется по их номерам.

ПРИМЕРЫ РАБОТЫ ФОРТ-СИСТЕМЫ

Рассмотрим следующий пример. Введем три новых слова:

```
: 3DUP   DUP 2OVER ROT ;
: S2     DUP × SWAP DUP × + ;
: SPD 3DUP S2 —ROT S2 × —ROT S2 × ;
```

Первые два слова нам уже встречались. Третье слово преобразует лежащие на стеке три числа, a, b, c в произведение попарных сумм квадратов (a×a+b×b)×(b×b+c×c)×(c×c+a×a).

При компиляции этих слов получается приблизительно такой текст (при его составлении использовалась шестнадцатиричная система счисления):

адрес	фрагмент текста	комментарий
		статья для 3DUP
1C12	04 33 44 55 50	строка '3DUP'
1C17	03 1C	1C03=адрес предыдущей статьи
1C19	CD F0 02	передача управления интерпретатору по адресу: 02F0 (режим CALL) CD — код команда передачи управления CALL
1C1C	1A 03	031A=адрес DUP
1C1E	78 03	0378=адрес 2OVER
1C20	5E 03	035E=адрес ROT
1C22	02 03	0302=адрес EXIT (режим RETURN)
		статья для S2
1C24	02 53 32	строка 'S2'
1C27	12 1C	1C12=адрес предыдущей статьи

1C29	CD F0 02	передача управления интерпретатору
1C2C	1A 03	031A=адрес DUP
1C2E	7F 03	037F=адрес ×
1C30	24 03	0324=адрес SWAP
1C32	1A 03	031A=адрес DUP
1C34	7F 03	037F=адрес ×
1C36	50 03	0350=адрес +
1C38	02 03	0302=адрес EXIT
		статья для SPD
1C3A	03 53 50 44	строка 'SPD'
1C3E	24 1C	1C24=адрес предыдущей статьи
1C40	CD F0 02	передача управления интерпретатору
1C43	19 1C	1C19=адрес 3DUP
1C45	29 1C	1C29=адрес S2
1C47	3E 03	033E=адрес —ROT
1C49	29 1C	1C29=адрес S2
1C4B	7F 03	037F=адрес ×
1C4D	3E 03	033E=адрес —ROT
1C4F	29 1C	1C29=адрес S2
1C51	7F 03	037F=адрес ×
1C53	02 03	0302=адрес EXIT

Посмотрим, как выполняется слово SPD, если в стеке уже лежат числа 5, 2 и 4. Нас будет интересовать содержимое основного стека, стека возвратов и указателя интерпретации IP, имеющее первоначально следующие значения:

основной стек	стек возвратов	IP
5 2 4		0C76

Начинается выполнение слова SPD. Форт-система передает управление программному тексту, расположенному по адресу 1C40. Указатель интерпретации IP содержит адрес 0C76. Это внутренний адрес самой системы, по которому следует вернуть управление после выполнения слова SPD. Управление получает подпрограмма адресного интерпретатора, реализующая режим CALL и располагающаяся по адресу 02F0. Эта подпрограмма переустанавливает IP на адрес 1C43, сохраняет прежнее значение IP в стеке возвратов и передает управление подпрограмме, реализующей режим NEXT:

5 2 4	0C76	1C43

Режим NEXT передает управление по адресу, на который указывает IP, т. е. по адресу 1C19. Одновременно IP увеличивается на 2:

5 2 4	0C76	1C45

Управление опять получит режим CALL, который произведет уже знакомые нам действия, и мы войдем в слово 3DUP:

5	2	4					0C76	1C45	1C1C

Режим NEXT передает управление по адресу 031A, где находится машинная программа, реализующая операцию DUP:

5	2	4	4				0C76	1C45	1C1E

Существует правило, по которому каждая машинная программа после своего выполнения передает управление режиму NEXT. Тем самым мы постоянно переходим к следующему по порядку слову:

5	2	4	4	5	2		0C76	1C45	1C20
5	2	4	5	2	4		0C76	1C45	1C22

Выполнение слова 3DUP завершается исполнением слова EXIT, которое, осуществляя режим RETURN, загружает IP значением, снятым со стека возвратов, и передает управление режиму NEXT:

5	2	4	5	2	4		0C76		1C45

Управление получит слово S2 и цикл повторится:

5	2	4	5	2	4		0C76	1C47	1C2C
5	2	4	5	2	4	4	0C76	1C47	1C2E
5	2	4	5	2	16		0C76	1C47	1C30
5	2	4	5	16	2		0C76	1C47	1C32
5	2	4	5	16	2	2	0C76	1C47	1C34
5	2	4	5	16	4		0C76	1C47	1C36
5	2	4	5	20			0C76	1C47	1C38
5	2	4	5	20			0C76		1C47

Выполнение S2 завершилось, выполняется —ROT:

5	2	20	4	5			0C76		1C49

Управление снова получает слово S2, но уже с другими параметрами:

5	2	20	4	5			0C76	1C4B	1C2C
5	2	20	4	5	5		0C76	1C4B	1C2E
5	2	20	4	25			0C76	1C4B	1C30
5	2	20	25	4			0C76	1C4B	1C32
5	2	20	25	4	4		0C76	1C4B	1C34
5	2	20	25	16			0C76	1C4B	1C36
5	2	20	41				0C76	1C4B	1C38
5	2	20	41				0C76		1C4B

Выполнение S2 завершилось, выполняются $\times$ и —ROT:

5	2	820					0C76		1C4D
820	5	2					0C76		1C4F

И снова управление получает слово S2:

820	5	2					0C76	1C51	1C2C

820 5	2	2	0C76	1C51	1C2E
820 5	4		0C76	1C51	1C30
820 4	5		0C76	1C51	1C32
820 4	5	5	0C76	1C51	1C34
820 4	25		0C76	1C51	1C36
820 29			0C76	1C51	1C38
820 29			0C76		1C51

Выполнение S2 завершилось, выполняются $\times$ и EXIT:

23780	0C76	1C53
23780		0C76

Слово SPD выполнено. Управление возвращается Форт-системе по адресу 0C76.

ПРИМЕР РАБОТЫ СТРУКТУР УПРАВЛЕНИЯ

Рассмотрим следующий пример:

: ABS DUP 0< IF NEGATE THEN ;

Это слово нам уже знакомо. Оно вычисляет абсолютную величину числа.

При компиляции слова ABS получится примерно следующий текст:

адрес	фрагмент текста				комментарий
1C55	03	41	42	53	строка 'ABS'
1C59	3A	1C			1C3A=адрес предыдущей статьи
1C5B	CD	F0	02		передача управления интерпретатору
1C5E	1A	03			031A=адрес DUP
1C60	BB	03			03BB=адрес 0<
1C62	12	04			0412=адрес ?BRANCH
1C64	68	1C			1C68=адрес перехода по FALSE
1C66	D1	03			03D1=адрес NEGATE
1C68	02	03			0302=адрес EXIT

Рассмотрим, как выполняется слово ABS, если на стеке находится число —5:

основной стек		стек возвратов	IP
—5			0C76
—5		0C76	1C5E
—5	—5	0C76	1C60
—5	—1	0C76	1C62

На стеке лежит значение *истина* и, следовательно, ?BRANCH передаст управление следующему слову:

—5	0C76	1C66

Выполняется NEGATE:

5	0C76	1C68
5		0C76

Если же на стеке будет лежать неотрицательное значение, то порядок выполнения слов изменится:

основной стек	стек возвратов	IP
4		0C76
4	0C76	1C5E
4 4	0C76	1C60
4 0	0C76	1C62

На стеке лежит значение *ложь* и, следовательно, ?BRANCH передаст управление по адресу, который записан вслед за этим словом. Таким образом, NEGATE не будет выполнено и управление получит сразу слово EXIT:

4	0C76	1C68
4		0C76

Мы подробно разобрали, как слово ABS выполняется. Посмотрим теперь, как оно компилируется, т. е. будем следить за тем, что делает Форт-система при обработке каждого из слов, входящих в определение ABS. Для этого нам придется следить за стеком, кодофайлом и указателем HERE:

слово	стек	HERE	кодофайл
		1C55	

Слово : создает заголовок словарной статьи с именем ABS :

: ABS		1C5E	03 41 42 53 3A 1C
			CD F0 02

Далее в кодофайл заносятся адреса слов DUP и 0<:

DUP		1C60	03 41 42 53 3A 1C
			CD F0 02 1A 03
0<		1C62	03 41 42 53 3A 1C
			CD F0 02 1A 03 BB 03

Слово IF исполняется во время компиляции, так как имеет признак немедленного исполнения:

IF	1C64 2	1C66	03 41 42 53 3A 1C CD
			F0 02 1A 03 BB 03 12
			04 00 00
NEGATE	1C64 2	1C68	03 41 42 53 3A 1C CD
			F0 02 1A 03 BB 03 12
			04 00 00 D1 03

Исполняется слово THEN. Значение HERE засылается по адресу, лежащему на стеке (раньше по этому адресу были нули):

THEN		1C68	03 41 42 53 3A 1C CD
			F0 02 1A 03 BB 03 12
			04 68 1C D1 03

Исполняется слово ; . Компилируется EXIT и создание статьи завершается:

;	1C6A	03	41	42	53	3A	1C	CD
		F0	02	1A	03	BB	03	12
		04	68	1C	D1	03	02	03

ПРИМЕР РАБОТЫ ОПРЕДЕЛЯЮЩИХ СЛОВ

На примере слова CONSTANT рассмотрим технику создания слов-генераторов. Это наиболее сложный пример. Он предполагает некоторое знакомство с машинными командами.

Итак, определим слово CONSTANT, при помощи которого можно задавать константы и с его помощью создадим две константы — TRUE и FALSE:

```
: CONSTANT CREATE , DOES> @ ;
—1 CONSTANT TRUE
0 CONSTANT FALSE
```

При компиляции слова ABS получится примерно следующий текст:

адрес	фрагмент текста	комментарий
	статья CONSTANT	
1C67	08 43 4F 4E 53 54 41 4E 54	строка 'CONSTANT'
1C70	55 1C	1C55=адрес предыдущей статьи
1C72	CD F0 02	передача управления интерпретатору
1C75	FA 12	12FA=адрес CREATE
1C77	50 10	1050=адрес ,
1C79	A0 14	14A0=адрес (DOES>)
1C7B	CD F0 02	передача управления интерпретатору текст A0 14 CD F0 02 компилируется словом DOES>
1C7E	50 04	0450=адрес @
1C80	02 03	0302=адрес EXIT
	статья TRUE	
1C82	04 54 52 55 45	строка 'TRUE'
1C87	67 1C	1C67=адрес предыдущей статьи
1C89	CD 78 1C	передача управления по адресу 1C7B
1C8B	FF FF	значение —1
	статья FALSE	

1C8D	05	46	41	4C	строка 'FALSE'
	53	45			
1C93	82	1C			1C82=адрес предыдущей статьи
1C95	CD	7B	1C		передача управления по адресу 1C7B
1C97	00	00			значение 0

Посмотрим, как выполняется слово TRUE:

основной стек	стек возвратов	IP
		0C76

Управление передается на адрес 1C89. По этому адресу стоит машинная команда передачи управления с возвратом CD 7B 1C. При своем исполнении эта команда положит на основной стек адрес возврата 1C8B и передаст управление на адрес 1C7B:

1C8B		0C76

Далее управление получит режим CALL адресного интерпретатора.

1C8B	0C76	1C7E

Выполнится слово @. По адресу 1C8B лежит —1:

—1	0C76	1C80

Произойдет возврат в Форт-систему по EXIT:

—1		0C76

Таким образом, мы на стеке получили значение —1.

Теперь посмотрим, как слова-генераторы создают новые слова, т. е. проследим, как выполняется текст:

—1 CONSTANT TRUE

стек	стек возвр.	IP	кодофайл
—1		0C76	

Управление получает слово CONSTANT по адресу 1C72:

—1	0C76	1C75

Слово CREATE создает заголовок словарной статьи с именем TRUE. Отметим, что слово : использует слово CREATE для создания заголовка:

—1	0C76	1C77	04 54 52 55 45 67
			1C CD F0 02

Компилируется значение константы:

—1	0C76	1C79	04 54 52 55 45 67
			1C CD F0 02 FF FF

Слово (DOES>) присваивает новое значение адресу интерпретатора (это адрес 1C7B, непосредственно следующий за

(DOES>) — на него указывает IP) и само выполняет EXIT, тем самым заканчивая исполнение слова CONSTANT:

```
—1                 0C76  04  54  52  55  45  67
                         1C  CD  7B  1C  FF  FF
```

Управление возвращено Форт-системе.

РЕАЛИЗАЦИЯ ЯЗЫКА ФОРТ

Форт с момента своего появления в начале семидесятых годов прошел большой путь от специализированного языка для управления радиотелескопом до универсального средства, на котором в настоящее время пишутся трансляторы и базы данных, видеоигры и экспертные системы. Особенно большое распространение Форт получил при решении задач реального времени; здесь оказались важными такие его характеристики, как высокая скорость, диалоговый режим работы и возможность настраиваться на работу с любым оборудованием.

В нашей стране ведутся активные работы по использованию языка Форт, охватывающие все перечисленные области его применения; созданы самостоятельные реализации языка на большинстве отечественных ЭВМ и серийно выпускаемых микропроцессоров. Создана аппаратная реализация языка Форт, представляющая собой специализированный процессор, непосредственно исполняющий фортовские слова как машинные команды. Такие Форт-машины очень эффективны. Их скорость достигает миллиона и более команд (таких как SWAP, +, CALL, EXIT и другие) в секунду.

Перечислим некоторые распространенные в нашей стране Форт-системы.

Единый Форт для К580, К1801, К1810. Система разработана С. Б. Кацевым, В. А. Кириллиным, А. А. Клубовичем и Н. Р. Ноздруновым (Ленинградский университет, НИИсчетмаш) и представляет собой единую реализацию языка Форт для микро-ЭВМ и микропроцессорных устройств на базе микропроцессоров серий К580, К1801, К1810 и специализированного высокопроизводительного Форт-процессора. Система установлена на следующие микро- и мини-ЭВМ: ЕС-7970, СМ-1800, «Роботрон», К1, КТС ЛИУС, КУВТ «Корвет», ПЭВМ «Сура», СМ-3, СМ-4, ДВК, «Электроника-60», ЕС1840.

Форт-ЕС. Система разработана С. Н. Барановым (Ленинградский институт информатики и автоматизации АН СССР — ЛИИАН) для ЕС-ЭВМ. Работает под управлением операционных систем ОС ЕС и СВМ ЕС.

Астро-форт. Система разработана И. Р. Агамирзяном и Г. М. Шуваловым (Институт теоретической астрономии АН СССР — ИТА АН СССР, ЛИИАН) для ЭВМ ЕС1840. Ее отличительной особенностью является поддержка многоэкранного режима работы.

Форт-«Искра-226». Система разработана А. Ю. Болдыревым, Н. Н. Веретеновой, Г. В. Лезиным, Е. Ф. Силиным (Институт социально-экономических проблем АН СССР) и представляет собой развитую операционную систему на базе языка Форт.

Форт-М6000. Система разработана В. Н. Патрышевым (Ленинград) для ЭВМ М6000. Работает как под управлением операционных систем ДОС РВ и РТЕ-2, так и без операционной системы.

Форт-БЭСМ-6. Система разработана И. Р. Агамирзяном (ИТА АН СССР, ЛИИАН) для ЭВМ БЭСМ-6. Работает под управлением операционной системы ДИСПАК.

Форт-«Эльбрус». Система разработана А. Е. Соловьевым (Ленинградский университет) для МВК «Эльбрус». Работает под управлением ОС «Эльбрус».

ПРИЛОЖЕНИЕ К. ТАБЛИЦА КОДОВ КОИ-8

В этой таблице приводятся символы, входящие в стандарт кодировки КОИ-8. Строки таблицы соответствуют левым половинам байтов изображенных символов, столбцы — правым. В шестнадцатиричной системе счисления, например, символ ? имеет код 3F.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2		!	″	#	¤	%	&	′	(	)	×	+	,	–	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	|	\	|	∧	—
6	′	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
C	ю	а	б	ц	д	е	ф	г	х	и	й	к	л	м	н	о
D	п	я	р	с	т	у	ж	в	ь	ы	з	ш	э	щ	ч	ь
E	Ю	А	Б	Ц	Д	Е	Ф	Г	Х	И	Й	К	Л	М	Н	О
F	П	Я	Р	С	Т	У	Ж	В	Ь	Ы	З	Ш	Э	Щ	Ч	

ЛИТЕРАТУРА

Астановский А. Г., Ломунов В. Н. Процессор, ориентированный на язык Форт. В сб.: Программирование микропроцессорной техники. — Таллинн: 1984, с. 50—67.

Баранов С. Н., Кириллин В. А., Ноздрунов Н. Р. Реализация языка Форт на дисплейном комплексе ЕС-7970. В сб.: Программирование микропроцессорной техники. — Таллинн: 1984, с. 41—49.

Баранов С. Н., Ноздрунов Н. Р. Язык Форт и его реализация. — Л.: 1988.

Вульф А. Операционные системы реального времени в русле развития вычислительной техники. — Электроника, 1985, № 17, с. 46—56.

ОГЛАВЛЕНИЕ

Стр.

Андрей Юрьевич БУРАГО,
Вячеслав Алексеевич КИРИЛЛИН,
Иосиф Владимирович РОМАНОВСКИЙ

Форт — язык для микропроцессоров

Научный редактор
кандидат технических наук Б. А. Кацев
Ответственный за выпуск

ст. референт Правления Ленинградской
организации общества «Знание» РСФСР
Т. В. Старостина
Редактор Л. В. Павлова
Обложка работы В. И. Меньшикова
Техн. редактор С. А. Кучерова
Корректор О. Г. Семенова

Сдано в набор 15.08.88 г. Подписано в печать 27.07.89 г. М-21210.
Формат 60×90[1]/16. Бумага тип. № 3. Гарнитура литературная.
Печать высокая. Усл. п. л. 2,25. Уч.-изд. л. 2,25. Тираж 26 000 экз.
Заказ № 3041. Цена 10 коп.

Ленинградская организация общества «Знание» РСФСР
191104, Ленинград, Литейный пр., 42.

Производственно-полиграфическое объединение № 1
Ленполиграфиздата. Пушкинское производство